A picture containing arrow

Description automatically generated

**ENGO 563 – Data Analysis in Engineering**

**Lab Report #1**

**Course Instructor:** Dr Yang Gao

**Teaching Assistant:** Wei Ding

**Due Date:** 09/30/21

**Date of Submission:** 09/30/21

**Student Name:** Jan Erik Naess (30068695)

Table of Contents

[Table of Figures 4](#_Toc83908228)

[Introduction 5](#_Toc83908229)

[Methodology 5](#_Toc83908230)

[Angular Model Partial Derivatives 6](#_Toc83908231)

[Distance Model Partial Derivatives 6](#_Toc83908232)

[Results and Analysis 7](#_Toc83908233)

[1. Estimate the coordinates of Points P1, P2 and their variance-covariance matrix 7](#_Toc83908234)

[2. Calculate the semi-major, semi-minor axis and the orientation of the standard error ellipse for points P1 and P2 8](#_Toc83908235)

[3. Calculate the adjusted observations and their variance-covariance matrix 9](#_Toc83908236)

[4. Calculate the observation residuals and their variance-covariance matrix 10](#_Toc83908237)

[5. Calculate the posteriori variance factor 10](#_Toc83908238)

[6. Are the results acceptable? Are the residuals acceptable? Justify. 11](#_Toc83908239)

[Discussion 12](#_Toc83908240)

[Angular Model vs. Directional Model 12](#_Toc83908241)

[Error Conversion Importance 13](#_Toc83908242)

[Conversion Criteria 13](#_Toc83908243)

[Automated output of results and matrices 13](#_Toc83908244)

[Conclusion 14](#_Toc83908245)

[References 14](#_Toc83908246)

[Appendices 14](#_Toc83908247)

[Cl Values 14](#_Toc83908248)

[Cr values 14](#_Toc83908249)

[Sample Design Matrix 16](#_Toc83908250)

[Github Repository 19](#_Toc83908251)

[Github Commits 20](#_Toc83908252)

[Tools.py 21](#_Toc83908253)

[Tables.py 23](#_Toc83908254)

[Net.py 26](#_Toc83908255)

[LeastSquares.py 40](#_Toc83908256)

[Distance.py 45](#_Toc83908257)

[Angle.py 55](#_Toc83908258)

[PostAdjustmentTester.py 70](#_Toc83908259)

Table of Equations

Equation 1: Distance Model 5

Equation 2: Angle Model 5

Equation 3: Tangent Derivative Expression 5

Equation 4: Angle Partial Derivative for Xi 6

Equation 5: Angle Partial Derivative for Yi 6

Equation 6: Angle Partial Derivative for Xj 6

Equation 7: Angle Partial Derivative for Yj 6

Equation 8: Angle Partial Derivative for Xk 6

Equation 9: Angle Partial Derivative for Yk 6

Equation 10: Distance Partial Derivative for Xi 6

Equation 11: Distance Partial Derivative for Yi 7

Equation 12: Distance Partial Derivative for Xj 7

Equation 13: Distance Partial Derivative for Yj 7

# Table of Figures

[Figure 1: Estimated coordinates of points P1, P2 7](#_Toc83907896)

[Figure 2: Covariance Matrix of Unknowns 8](#_Toc83907897)

[Figure 3: Covariance Matrix of Unknowns 8](#_Toc83907898)

[Figure 4: Standard Error Ellipses 8](#_Toc83907899)

[Figure 5: Adjusted Observations (m and rad) 9](#_Toc83907900)

[Figure 6: Covariance Matrix of Adjusted Observations (see appendix for numerical values) 9](#_Toc83907901)

[Figure 7: Observation Residuals 10](#_Toc83907902)

[Figure 8: Covariance Matrix of Residuals (See appendix for numerical values) 10](#_Toc83907903)

[Figure 9: A-Posteriori Variance Factor 10](#_Toc83907904)

[Figure 10: Global A-Posteriori Variance Factor Test 11](#_Toc83907905)

[Figure 11: Significance test on estimated parameters 11](#_Toc83907906)

[Figure 12: Goodness-of-fit test on residuals 11](#_Toc83907907)

[Figure 13: Blunder Test 12](#_Toc83907908)

# Introduction

This lab was completed with python scripts that were designed to conduct least squares adjustments on a geodetic network and to help analyze results. The survey was comprised of 12 angle and 6 distance measurements with respective precisions of . All measurements were input as uncorrelated and of the five points, three were known (A, B, C) and two were unknowns (P1 and P2).

Software was developed within Python using common libraries such as math, numpy and pandas. Data was visualized using matplotlib and was processed with four in-house classes. The “Network” class conducted the overarching LSA. The two functional model classes, “Angle” and “Distance” read in data and completed iteration updated when requested. The “LSA” super class was inherited by all classes and contained metrics such as and helped with sorting of columns to correctly input and update important matrices.

# Methodology

Two types of observations were observed: distance and angle. From here it was determined that a 2D distance functions model and an angular function model would be used to complete the least squares adjustment. The equations for these two respective models may be seen below.

Equation 1: Distance Model

Equation 2: Angle Model

Equation 3: Tangent Derivative Expression

The measurements’ errors were read in as arc seconds and meters and converted to error of radians and meters. This was to facilitate the mathematical operations that Python can best conduct. The a-priori variance factor was then set to be the inverse of the angular error as a way of making the weight matrix more understandable. This resulting in an a-priori variance factor of: .

The partial derivatives of all components then needed to be derived to populate the Design Matrix. Variables with the subscript *k* represent the coordinates of the point in which the bearing was taken **from**. Variables with the subscript *i* represent the coordinates of the point in which the angle or distance measurement was taken **at**. Variables with the subscript *j* represent the coordinates of the point in which the measurement was taken **to**.

## Angular Model Partial Derivatives (unitless)

Equation 4: Angle Partial Derivative for Xi

Equation 5: Angle Partial Derivative for Yi

Equation 6: Angle Partial Derivative for Xj

Equation 7: Angle Partial Derivative for Yj

Equation 8: Angle Partial Derivative for Xk

Equation 9: Angle Partial Derivative for Yk

## Distance Model Partial Derivatives (1/m)

Equation 10: Distance Partial Derivative for Xi

Equation 11: Distance Partial Derivative for Yi

Equation 12: Distance Partial Derivative for Xj

Equation 13: Distance Partial Derivative for Yj

When constructing the Design matrix, it should be noted that columns were only made for the unknown’s parameters of X and Y. This is because the partial derivate of any known is zero.

# Results and Analysis

## Estimate the coordinates of Points P1, P2 and their variance-covariance matrix (m)

Text

Description automatically generated

Figure 1: Estimated coordinates of points P1, P2

Chart, treemap chart

Description automatically generated

Figure 2: Covariance Matrix of Unknowns

Letter

Description automatically generated

Figure 3: Covariance Matrix of Unknowns

## Calculate the semi-major, semi-minor axis and the orientation of the standard error ellipse for points P1 and P2

Text, letter

Description automatically generated

Figure 4: Standard Error Ellipses

## Calculate the adjusted observations and their variance-covariance matrix

Table

Description automatically generated with low confidence

Figure 5: Adjusted Observations (m and radians)

A picture containing table

Description automatically generated

Figure 6: Covariance Matrix of Adjusted Observations (see appendix for numerical values)

It should be noted that the values in here are rounded to the first decimal. This is simply because of an error with outputting the image version of the matrix. Numerical values may be referenced in the reference. The variance observed in the distance observations may be seen in the coloring of the first six rows.

## Calculate the observation residuals and their variance-covariance matrix

Table

Description automatically generated with medium confidence

Figure 7: Observation Residuals (m and radians)

A picture containing table

Description automatically generated

Figure 8: Covariance Matrix of Residuals (See appendix for numerical values)

Higher levels of variance for the first six rows is a result of lower quality distance measurements in comparison to the precision of the angular measurements. Thus, it was to be expected that poorer quality measurements will display higher standard deviations from the best estimated value.

## Calculate the posteriori variance factor

Graphical user interface, text, application, email

Description automatically generated

Figure 9: A-Posteriori Variance Factor(unitless)

## Are the results acceptable? Are the residuals acceptable? Justify.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 10: Global A-Posteriori Variance Factor Test

The a-posteriori variance factor test failed indicating that there may have been errors or poorly chosen math models. The failure of this test is a strong enough indication that other components should be tested but is not a guarantee that there are any major issues.

Graphical user interface

Description automatically generated with low confidence

Figure 11: Significance test on estimated parameters

The parameters of the coordinates for P1 and P2 were then checked. These both passed at the 95% confidence level and could be safely accepted as good final results.

A picture containing text

Description automatically generated

Figure 12: Goodness-of-fit test on residuals

A semi-global test was then conducted on the residuals which passed. This indicated that there were no major errors detectable at initial glance of the results.

Table

Description automatically generated

Figure 13: Blunder Test

The residuals were then more thoroughly reviewed on a local level to the 99% confidence level. All residuals passed indicating that there were most likely no blunders. Therefore, the Global A-Posteriori test may have failed due to observations that were not well paired with each other. The initial apriori was set to be the inverse of 1.5 arc seconds when converted to radians. The 1.5” error was far more precise than the two-centimeter error from the distance observations. But because the distance observations had a relatively lower level of precision, they may have more been as beneficial to the overall network. Additionally, the angular functional model will have a larger level of correlation between results than if it were to be switched to the directional functional model.

# Discussion

## Angular Model vs. Directional Model

As previously mentioned, the angular model is based on the bearings taken from the points measured. When the point that the measurement is taken from remain constant, then little correlation occurs. But when one of the points has a bearing measured from an unknown, then correlation begins to occur. This is seen in when the unknown point’s coordinate change, then the bearing shifts serval other points with that initial change. This correlation has the potential of being reduced by implementing a directional model. In order to implement a directional model, a nuisance angle is added to each bearing to make the bearing originate from an obituary north. All angle observations are then observed from a ‘known’ point/direction which reduces the correlation when points begin being corrected.

## Error Conversion Importance

It is common practice in surveying to provide angular precision in arc seconds and distance precision in whole numbers (2cm instead of .02m). There are two important adjustments that must made to errors in order to integrate them into a programable least-squares-adjustment. The first adjustment is to the angular error, which must be converted into decimal degrees, and is often then converted from degrees to radians. Radians is preferred because most code libraries by default read and return angle measurements in radians. The second adjustment that should be made is in the conversion of non-meter errors into decimal meters. This is because measurements and positioning are normally provided in meters. If they are provided in a different unit, then all distance and x, y, z coordinates and errors should at the bare minimum be uniform. Otherwise, disproportional adjustments will occur.

## Conversion Criteria

It is a commonly accepted practice to have the conversion criteria of an LSA be based off of the parameter’s standard deviations. This means that Cx is computed and then the diagonal elements are taken out and square rooted to see their standard deviation. Once all standard deviations are below one-half of the observation’s standard deviations then it is often acceptable to end convergence. Because the functional models used were well suited for this application, a simpler convergence criterion was used. This LSA was programmed to meet convergence at .0001 meters: of which was converged to after the second iterations.

## Automated output of results and matrices

Least-squares-adjustments often use repeatable statistical tests and desire similar formats of outputted results. Additional time was invested in this lab to create several classes for performing fully autonomous analysis and matrix figure generation. The “Tools” class was created to automate visualization of importance matrices. The “PostAdjustmentTester” class was created to leverage final matrices outputs and conduct statistical tests on them. Lastly, the “Tables” class was to consolidate all statistic table values in an easily accessible and formattable location so that statistical tests could be easily conducted and automated.

# Conclusion

Overall, we were able to reach our desired results for the coordinates of points P1 and P2 as seen in Figure 1. Their error ellipses had similar magnitudes of minor and major axis indicated a well-constructed network (thus circular ellipses) and were both in .2 to .04 meters range indicating a moderate level of precision extracted from this set of observations. Statistical tests indicated that there were no outliers and therefore no reason to eliminate any of the measurements. To better determine the quality of the results, one must better know the application of these coordinates.

# References

Gao, Y. (2021). *ENGO563 Lab#1 - Fall 2021.* Retrieved from D2L: <https://d2l.ucalgary.ca/d2l/le/content/399854/viewContent/4858929/View>

Gao, Y. (2021). *Lab1 - Instructions.* Retrieved from D2L: <https://d2l.ucalgary.ca/d2l/le/content/399854/viewContent/4877097/View>

El-Sheimy, N. (2021). *Review of least squares (parametric).* Retrieved from D2L: <https://d2l.ucalgary.ca/d2l/le/content/399854/viewContent/4843398/View>

Detchev, I. (2020). *Examples for Post-Adjustment Tests.* [PDF]

# Appendices

## Cl Values (

matrix([[ 9.16221828e-04, 5.19493709e-05, -7.71080943e-04,

-3.09145239e-04, 3.78451671e-06, 9.58335007e-05,

-2.51129663e-07, 2.68177079e-07, -1.70474162e-08,

-2.68177079e-07, 3.63391280e-07, -9.52142010e-08,

-4.42148501e-08, -1.38485303e-07, 1.82700153e-07,

-1.27823055e-07, 2.62236855e-08, 1.01599369e-07],

[ 5.19493709e-05, 7.54796094e-04, -4.23460478e-04,

3.86253617e-04, 2.67394089e-04, -3.18988940e-04,

-2.98344228e-07, -7.43938571e-09, 3.05783613e-07,

7.43938571e-09, -1.27324192e-07, 1.19884807e-07,

-2.35260966e-07, 2.25283166e-07, 9.97780047e-09,

-1.17838780e-07, 2.00385254e-07, -8.25464738e-08],

[-7.71080943e-04, -4.23460478e-04, 8.40729415e-04,

5.62328210e-05, -1.38130549e-04, 8.32053111e-05,

3.54841939e-07, -2.14257144e-07, -1.40584795e-07,

2.14257144e-07, -2.31110786e-07, 1.68536417e-08,

1.54768773e-07, -1.20313058e-09, -1.53565643e-07,

1.63431126e-07, -1.22528022e-07, -4.09031036e-08],

[-3.09145239e-04, 3.86253617e-04, 5.62328210e-05,

2.71609294e-03, 2.15960767e-05, 8.33527225e-04,

-6.78445570e-08, -1.02647924e-07, 1.70492481e-07,

1.02647924e-07, -2.02058007e-07, 9.94100825e-08,

3.66533156e-07, -2.65237936e-07, -1.01295219e-07,

-3.88764944e-07, 5.35140500e-07, -1.46375556e-07],

[ 3.78451671e-06, 2.67394089e-04, -1.38130549e-04,

2.15960767e-05, 2.50130556e-03, 1.81979915e-03,

-1.01997675e-07, -6.93942719e-09, 1.08937102e-07,

6.93942719e-09, -5.10671648e-08, 4.41277376e-08,

-1.59997492e-07, 3.78116848e-07, -2.18119355e-07,

-1.46956500e-07, -2.25052008e-07, 3.72008508e-07],

[ 9.58335007e-05, -3.18988940e-04, 8.32053111e-05,

8.33527225e-04, 1.81979915e-03, 2.24672077e-03,

9.63238444e-08, 3.78216069e-08, -1.34145451e-07,

-3.78216069e-08, 1.01840315e-07, -6.40187082e-08,

2.58839188e-07, -7.82546817e-08, -1.80584506e-07,

-2.31640717e-07, -1.19909478e-07, 3.51550195e-07],

[-2.51129663e-07, -2.98344228e-07, 3.54841939e-07,

-6.78445570e-08, -1.01997675e-07, 9.63238444e-08,

1.76188983e-10, -6.49484278e-11, -1.11240555e-10,

6.49484278e-11, -4.37045136e-11, -2.12439141e-11,

1.00070806e-10, -5.01380904e-11, -4.99327155e-11,

7.68249978e-11, -8.23463786e-11, 5.52138079e-12],

[ 2.68177079e-07, -7.43938571e-09, -2.14257144e-07,

-1.02647924e-07, -6.93942719e-09, 3.78216069e-08,

-6.49484278e-11, 7.91771630e-11, -1.42287353e-11,

-7.91771630e-11, 1.10819648e-10, -3.16424850e-11,

-5.93134313e-12, -4.75562640e-11, 5.34876071e-11,

-3.40827751e-11, 1.68504370e-12, 3.23977314e-11],

[-1.70474162e-08, 3.05783613e-07, -1.40584795e-07,

1.70492481e-07, 1.08937102e-07, -1.34145451e-07,

-1.11240555e-10, -1.42287353e-11, 1.25469290e-10,

1.42287353e-11, -6.71151345e-11, 5.28863992e-11,

-9.41394628e-11, 9.76943544e-11, -3.55489161e-12,

-4.27422227e-11, 8.06613349e-11, -3.79191122e-11],

[-2.68177079e-07, 7.43938571e-09, 2.14257144e-07,

1.02647924e-07, 6.93942719e-09, -3.78216069e-08,

6.49484278e-11, -7.91771630e-11, 1.42287353e-11,

7.91771630e-11, -1.10819648e-10, 3.16424850e-11,

5.93134313e-12, 4.75562640e-11, -5.34876071e-11,

3.40827751e-11, -1.68504370e-12, -3.23977314e-11],

[ 3.63391280e-07, -1.27324192e-07, -2.31110786e-07,

-2.02058007e-07, -5.10671648e-08, 1.01840315e-07,

-4.37045136e-11, 1.10819648e-10, -6.71151345e-11,

-1.10819648e-10, 1.73233236e-10, -6.24135881e-11,

2.82584169e-11, -1.00795833e-10, 7.25374160e-11,

-2.89380090e-11, -2.87328897e-11, 5.76708987e-11],

[-9.52142010e-08, 1.19884807e-07, 1.68536417e-08,

9.94100825e-08, 4.41277376e-08, -6.40187082e-08,

-2.12439141e-11, -3.16424850e-11, 5.28863992e-11,

3.16424850e-11, -6.24135881e-11, 3.07711031e-11,

-3.41897600e-11, 5.32395689e-11, -1.90498088e-11,

-5.14476615e-12, 3.04179334e-11, -2.52731673e-11],

[-4.42148501e-08, -2.35260966e-07, 1.54768773e-07,

3.66533156e-07, -1.59997492e-07, 2.58839188e-07,

1.00070806e-10, -5.93134313e-12, -9.41394628e-11,

5.93134313e-12, 2.82584169e-11, -3.41897600e-11,

1.70214318e-10, -1.58548572e-10, -1.16657460e-11,

-3.09417392e-11, 3.02193489e-11, 7.22390253e-13],

[-1.38485303e-07, 2.25283166e-07, -1.20313058e-09,

-2.65237936e-07, 3.78116848e-07, -7.82546817e-08,

-5.01380904e-11, -4.75562640e-11, 9.76943544e-11,

4.75562640e-11, -1.00795833e-10, 5.32395689e-11,

-1.58548572e-10, 2.04261741e-10, -4.57131690e-11,

3.86626380e-11, -5.33278175e-11, 1.46651795e-11],

[ 1.82700153e-07, 9.97780047e-09, -1.53565643e-07,

-1.01295219e-07, -2.18119355e-07, -1.80584506e-07,

-4.99327155e-11, 5.34876071e-11, -3.55489161e-12,

-5.34876071e-11, 7.25374160e-11, -1.90498088e-11,

-1.16657460e-11, -4.57131690e-11, 5.73789151e-11,

-7.72089886e-12, 2.31084686e-11, -1.53875697e-11],

[-1.27823055e-07, -1.17838780e-07, 1.63431126e-07,

-3.88764944e-07, -1.46956500e-07, -2.31640717e-07,

7.68249978e-11, -3.40827751e-11, -4.27422227e-11,

3.40827751e-11, -2.89380090e-11, -5.14476615e-12,

-3.09417392e-11, 3.86626380e-11, -7.72089886e-12,

9.86401715e-11, -8.65496269e-11, -1.20905446e-11],

[ 2.62236855e-08, 2.00385254e-07, -1.22528022e-07,

5.35140500e-07, -2.25052008e-07, -1.19909478e-07,

-8.23463786e-11, 1.68504370e-12, 8.06613349e-11,

-1.68504370e-12, -2.87328897e-11, 3.04179334e-11,

3.02193489e-11, -5.33278175e-11, 2.31084686e-11,

-8.65496269e-11, 1.64407086e-10, -7.78574590e-11],

[ 1.01599369e-07, -8.25464738e-08, -4.09031036e-08,

-1.46375556e-07, 3.72008508e-07, 3.51550195e-07,

5.52138079e-12, 3.23977314e-11, -3.79191122e-11,

-3.23977314e-11, 5.76708987e-11, -2.52731673e-11,

7.22390253e-13, 1.46651795e-11, -1.53875697e-11,

-1.20905446e-11, -7.78574590e-11, 8.99480036e-11]])

## Cr values

matrix([[ 4.31740263e-03, -5.19493709e-05, 7.71080943e-04,

3.09145239e-04, -3.78451671e-06, -9.58335007e-05,

2.51129663e-07, -2.68177079e-07, 1.70474162e-08,

2.68177079e-07, -3.63391280e-07, 9.52142010e-08,

4.42148501e-08, 1.38485303e-07, -1.82700153e-07,

1.27823055e-07, -2.62236855e-08, -1.01599369e-07],

[-5.19493709e-05, 4.47882836e-03, 4.23460478e-04,

-3.86253617e-04, -2.67394089e-04, 3.18988940e-04,

2.98344228e-07, 7.43938571e-09, -3.05783613e-07,

-7.43938571e-09, 1.27324192e-07, -1.19884807e-07,

2.35260966e-07, -2.25283166e-07, -9.97780047e-09,

1.17838780e-07, -2.00385254e-07, 8.25464738e-08],

[ 7.71080943e-04, 4.23460478e-04, 4.39289504e-03,

-5.62328210e-05, 1.38130549e-04, -8.32053111e-05,

-3.54841939e-07, 2.14257144e-07, 1.40584795e-07,

-2.14257144e-07, 2.31110786e-07, -1.68536417e-08,

-1.54768773e-07, 1.20313058e-09, 1.53565643e-07,

-1.63431126e-07, 1.22528022e-07, 4.09031036e-08],

[ 3.09145239e-04, -3.86253617e-04, -5.62328210e-05,

2.51753152e-03, -2.15960767e-05, -8.33527225e-04,

6.78445570e-08, 1.02647924e-07, -1.70492481e-07,

-1.02647924e-07, 2.02058007e-07, -9.94100825e-08,

-3.66533156e-07, 2.65237936e-07, 1.01295219e-07,

3.88764944e-07, -5.35140500e-07, 1.46375556e-07],

[-3.78451671e-06, -2.67394089e-04, 1.38130549e-04,

-2.15960767e-05, 2.73231889e-03, -1.81979915e-03,

1.01997675e-07, 6.93942719e-09, -1.08937102e-07,

-6.93942719e-09, 5.10671648e-08, -4.41277376e-08,

1.59997492e-07, -3.78116848e-07, 2.18119355e-07,

1.46956500e-07, 2.25052008e-07, -3.72008508e-07],

[-9.58335007e-05, 3.18988940e-04, -8.32053111e-05,

-8.33527225e-04, -1.81979915e-03, 2.98690369e-03,

-9.63238444e-08, -3.78216069e-08, 1.34145451e-07,

3.78216069e-08, -1.01840315e-07, 6.40187082e-08,

-2.58839188e-07, 7.82546817e-08, 1.80584506e-07,

2.31640717e-07, 1.19909478e-07, -3.51550195e-07],

[ 2.51129663e-07, 2.98344228e-07, -3.54841939e-07,

6.78445570e-08, 1.01997675e-07, -9.63238444e-08,

5.15761181e-10, 6.49484278e-11, 1.11240555e-10,

-6.49484278e-11, 4.37045136e-11, 2.12439141e-11,

-1.00070806e-10, 5.01380904e-11, 4.99327155e-11,

-7.68249978e-11, 8.23463786e-11, -5.52138079e-12],

[-2.68177079e-07, 7.43938571e-09, 2.14257144e-07,

1.02647924e-07, 6.93942719e-09, -3.78216069e-08,

6.49484278e-11, 6.12773001e-10, 1.42287353e-11,

7.91771630e-11, -1.10819648e-10, 3.16424850e-11,

5.93134313e-12, 4.75562640e-11, -5.34876071e-11,

3.40827751e-11, -1.68504370e-12, -3.23977314e-11],

[ 1.70474162e-08, -3.05783613e-07, 1.40584795e-07,

-1.70492481e-07, -1.08937102e-07, 1.34145451e-07,

1.11240555e-10, 1.42287353e-11, 5.66480874e-10,

-1.42287353e-11, 6.71151345e-11, -5.28863992e-11,

9.41394628e-11, -9.76943544e-11, 3.55489161e-12,

4.27422227e-11, -8.06613349e-11, 3.79191122e-11],

[ 2.68177079e-07, -7.43938571e-09, -2.14257144e-07,

-1.02647924e-07, -6.93942719e-09, 3.78216069e-08,

-6.49484278e-11, 7.91771630e-11, -1.42287353e-11,

6.12773001e-10, 1.10819648e-10, -3.16424850e-11,

-5.93134313e-12, -4.75562640e-11, 5.34876071e-11,

-3.40827751e-11, 1.68504370e-12, 3.23977314e-11],

[-3.63391280e-07, 1.27324192e-07, 2.31110786e-07,

2.02058007e-07, 5.10671648e-08, -1.01840315e-07,

4.37045136e-11, -1.10819648e-10, 6.71151345e-11,

1.10819648e-10, 5.18716928e-10, 6.24135881e-11,

-2.82584169e-11, 1.00795833e-10, -7.25374160e-11,

2.89380090e-11, 2.87328897e-11, -5.76708987e-11],

[ 9.52142010e-08, -1.19884807e-07, -1.68536417e-08,

-9.94100825e-08, -4.41277376e-08, 6.40187082e-08,

2.12439141e-11, 3.16424850e-11, -5.28863992e-11,

-3.16424850e-11, 6.24135881e-11, 6.61179061e-10,

3.41897600e-11, -5.32395689e-11, 1.90498088e-11,

5.14476615e-12, -3.04179334e-11, 2.52731673e-11],

[ 4.42148501e-08, 2.35260966e-07, -1.54768773e-07,

-3.66533156e-07, 1.59997492e-07, -2.58839188e-07,

-1.00070806e-10, 5.93134313e-12, 9.41394628e-11,

-5.93134313e-12, -2.82584169e-11, 3.41897600e-11,

5.21735846e-10, 1.58548572e-10, 1.16657460e-11,

3.09417392e-11, -3.02193489e-11, -7.22390253e-13],

[ 1.38485303e-07, -2.25283166e-07, 1.20313058e-09,

2.65237936e-07, -3.78116848e-07, 7.82546817e-08,

5.01380904e-11, 4.75562640e-11, -9.76943544e-11,

-4.75562640e-11, 1.00795833e-10, -5.32395689e-11,

1.58548572e-10, 4.87688423e-10, 4.57131690e-11,

-3.86626380e-11, 5.33278175e-11, -1.46651795e-11],

[-1.82700153e-07, -9.97780047e-09, 1.53565643e-07,

1.01295219e-07, 2.18119355e-07, 1.80584506e-07,

4.99327155e-11, -5.34876071e-11, 3.55489161e-12,

5.34876071e-11, -7.25374160e-11, 1.90498088e-11,

1.16657460e-11, 4.57131690e-11, 6.34571249e-10,

7.72089886e-12, -2.31084686e-11, 1.53875697e-11],

[ 1.27823055e-07, 1.17838780e-07, -1.63431126e-07,

3.88764944e-07, 1.46956500e-07, 2.31640717e-07,

-7.68249978e-11, 3.40827751e-11, 4.27422227e-11,

-3.40827751e-11, 2.89380090e-11, 5.14476615e-12,

3.09417392e-11, -3.86626380e-11, 7.72089886e-12,

5.93309992e-10, 8.65496269e-11, 1.20905446e-11],

[-2.62236855e-08, -2.00385254e-07, 1.22528022e-07,

-5.35140500e-07, 2.25052008e-07, 1.19909478e-07,

8.23463786e-11, -1.68504370e-12, -8.06613349e-11,

1.68504370e-12, 2.87328897e-11, -3.04179334e-11,

-3.02193489e-11, 5.33278175e-11, -2.31084686e-11,

8.65496269e-11, 5.27543078e-10, 7.78574590e-11],

[-1.01599369e-07, 8.25464738e-08, 4.09031036e-08,

1.46375556e-07, -3.72008508e-07, -3.51550195e-07,

-5.52138079e-12, -3.23977314e-11, 3.79191122e-11,

3.23977314e-11, -5.76708987e-11, 2.52731673e-11,

-7.22390253e-13, -1.46651795e-11, 1.53875697e-11,

1.20905446e-11, 7.78574590e-11, 6.02002160e-10]])

## Sample Design Matrix (1/m and unitless)

Chart, treemap chart

Description automatically generated

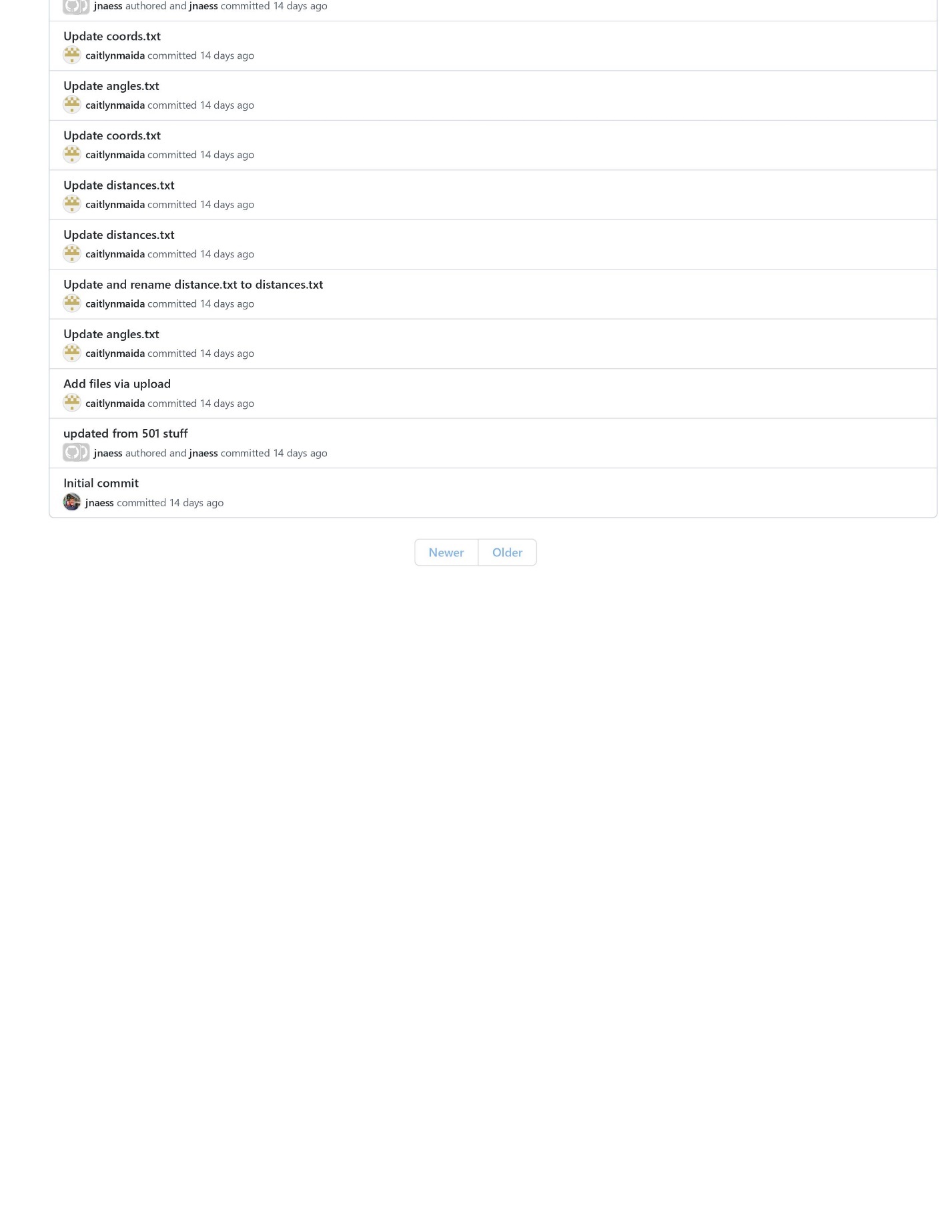
## Github Repository

https://github.com/jnaess/ENGO563.git

## Github Commits

Text

Description automatically generated



## Tools.py

import numpy as np

import matplotlib.pyplot as plt

import os

class Tools():

"""

Desc:

This class was made as a toolbox for plotting and converting values

"""

def \_\_init\_\_(self):

"""

Just exsists :-)

"""

def plot\_mat(self, matrix, title = "Title", round\_to = 6):

"""

Desc:

Checks to see if a "Figures" folder has been made. If it is not made then it makes it.

Then saves the input matrix as a .png to the folder with "Title" as the name

Input:

matrix: the numpy array to plot

title: the title of the array and output image (default "Title")

round\_to: decimals to round to (default 6)

Output:

"""

#set up figure with decently sized boxes

fig, ax = plt.subplots(figsize = (10,15))

ax.imshow(matrix)

plt.title(title)

# Loop over data dimensions and create text annotations.

for i in range(matrix.shape[0]):

for j in range(matrix.shape[1]):

#inputs numerical values

text = ax.text(j, i, round(matrix[i, j],round\_to),

ha="center", va="center", color="w")

#plt.axis('off')

#folder is just called figures

folder\_path = 'Figures/'

file\_name = title

#makes folder if not already there

if not os.path.isdir(folder\_path):

os.makedirs(folder\_path)

#saves to the folder using the title name

fig.savefig(os.path.join(folder\_path,file\_name))

plt.figure().clear()

plt.close()

plt.cla()

plt.clf()

## Tables.py

from scipy import misc

from scipy import stats

import pandas as pd

import numpy as np

class Tables():

"""

Parent class to PostAdjustmentTester which generates the significant values to increase modularity

"""

def \_\_init\_\_(self):

"""

"""

def newtons\_method(self, x, tolerance=0.0001):

while True:

x1 = x - self.f(x) / misc.derivative(self.f, x)

t = abs(x1 - x)

if t < tolerance:

break

x = x1

return x

def f(self, x):

return 1 - stats.chi2.cdf(x, self.r) - self.pvalue

def x\_2(self):

"""

Reference:

Code reformatted to return a single line of the desired x\_2 value based on our DOF (instead of a given value)

Code refers to functions "newtons\_method", "f", "x\_2"

https://moonbooks.org/Articles/How-to-create-a-Chi-square-table-using-python-/

Desc:

returns a chi-square dataframe row for the designated DOF

Input:

r: defrees of freedom

Output:

"""

self.pvalueList = [0.995, 0.99, 0.975, 0.95, 0.90, 0.10, 0.05, 0.025, 0.01, 0.005]

results = []

for i in range(self.r,self.r+1):

self.r = i

Result = []

for self.pvalue in self.pvalueList:

x0 = self.r # x0 approximation

x = self.newtons\_method(x0)

Result.append(x)

for i in range(10):

Result[i] = round(Result[i],3)

results.append(Result)

return pd.DataFrame(results, columns = self.pvalueList)

## Net.py

from numpy import transpose as t

from numpy import matrix as mat, matmul as mm

from numpy import linalg as lin

from numpy.linalg import inv

import math as m

import numpy as np

import pandas as pd

from LeastSquares import LS

from Level import Delta

from PostAdjustmentTester import PostAdjustmentTester

class Network(LS, PostAdjustmentTester):

"""

Build to run the least squares adjustment and set up the overall network

"""

def \_\_init\_\_(self, models):

"""

Desc:

Input:

models: list of models that have been initialized with

data. Must contain the same number of columns in their a

matrix (predefined by LS())

Output:

"""

LS.\_\_init\_\_(self)

PostAdjustmentTester.\_\_init\_\_(self)

self.models = models

self.initialize\_variables()

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_begin LSA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.nonlinear\_LSA()

def initialize\_variables(self):

"""

Desc:

initializes major variables (combining matrices and stuff)

Input:

Output:

self.u

"""

self.u = self.models[0].u

#set up observation matrix

temp = []

for obs in self.models:

temp.append(obs.obs)

self.obs = np.vstack(temp)

#set up errors matrix

temp = []

for obs in self.models:

temp.append(obs.errs)

self.errs = np.vstack(temp)

#set up number of observations variable

self.n = len(self.errs)

#set up design matrix

self.design()

#set up covariance (no additional formatting needed)

self.covariance()

#set up apriori

self.apriori = m.radians(1.5/3600)

#set up weight matrix

self.P = self.apriori\*\*2 \* inv(self.Cl)

def final\_matrices(self):

"""

Desc:

Once the LSA is completed then this generates all desired matrices for analysis

Input:

Output:

self.r\_hat: residuals

self.l\_hat: adjusted observations

self.a\_post: a-posteriori variance factor

self.uvf: unit variance factor

self.Cx (also Cs):

self.Cl:

self.Cr:

"""

self.r\_hat = mm(self.A,self.S\_hat) + self.w\_0

self.l\_hat = self.obs + self.r\_hat

self.a\_post = m.sqrt(mm(t(self.r\_hat),mm(self.P,self.r\_hat)/(self.n-self.u))[0,0])

self.uvf = self.a\_post\*\*2 / self.apriori\*\*2

self.Cx = self.a\_post\*\*2 \* inv(mm(t(self.A),mm(self.P,self.A)))

self.plot\_mat(self.Cx, "Covariance Matrix of Unknowns")

self.Cl = mm(self.A,mm(self.Cx,t(self.A)))

self.plot\_mat(self.Cl, "Covariance Matrix of Measurements")

self.Cr = self.a\_post\*\*2\*inv(self.P)-self.Cl

self.plot\_mat(self.Cr, "Covariance Matrix of Residuals")

def nonlinear\_LSA(self):

"""

Desc:

Iterates a nonlinear LSA, checking whether criterea was met. Once it was met then it constructs the final matrices for analysis

Input:

Output:

"""

self.not\_met = True

i = 0

self.w\_0 = mat(np.zeros((self.n, 1)))

self.S\_hat = mat(np.zeros((self.n, 1)))

self.x\_hat = mat(np.zeros((self.n, 1)))

while self.not\_met:

i = i + 1

#print("LSA iteration: " + str(i))

#print("x\_0: ")

#print(LS.x\_0)

#l\_0

self.obs\_0()

#update l\_0 and A

self.update\_values()

#misclosure

self.w\_0 = self.l\_0 - self.obs

#S\_hat

self.S\_hat = -mm(inv(mm(t(self.A),mm(self.P,self.A))),mm(t(self.A),mm(self.P,self.w\_0)))

#print("l\_0: ")

#print(self.l\_0)

#x\_hat

self.x\_hat = LS.x\_0 + self.S\_hat

#update x\_0

LS.x\_0 = self.x\_hat

#print("S\_hat:")

#print(self.S\_hat)

#print("x\_hat: ")

#print(self.x\_hat)

#print("A: ")

#print(self.A)

self.convergence(i)

#print("LSA passed in: " + str(i) + " iterations")

self.final\_matrices()

def error\_ellipses(self):

"""

Desc:

generates the error ellipses, minor, major, bearing\_major

\*\*must already have self.Cx generates\*\*

\*\*assumes Xa, Ya, Xb, Yb, Xc, Yc etc in the Cx diagonal\*\*

Input:

Output:

"""

self.u

ellipses = []

for i in range(0,self.Cx.shape[0],2):

q11 = self.Cx[i,i]

q12 = self.Cx[i,i+1]

q21 = self.Cx[i+1,i]

q22 = self.Cx[i+1,i+1]

ellipses.append(self.ellipse(q11, q12, q21, q22))

return ellipses

def ellipse(self, q11, q12, q21, q22):

"""

Desc:

Calculates the error ellipse, returns back a dataframe of the values

Input:

q11,

q12,

q21,

q22

Output:

{

"minor": float,

"major": float,

"major\_orientation": radians

}

"""

minor = m.sqrt(abs((q11 + q22 - m.sqrt((q11-q22)\*\*2+4\*(q12\*\*2)))/2))

major = m.sqrt(abs((q11 + q22 + m.sqrt((q11-q22)\*\*2+4\*(q12\*\*2)))/2))

major\_orientation = m.atan(q12/(major\*\*2-q22))

return {

"minor": minor,

"major": major,

"major\_orientation": major\_orientation

}

def convergence(self,i):

"""

Desc:

Checks based on this criterea, if convergence is met then sets self.not\_met to False

Input:

i: number of iterations (for simple # of ter break)

Output:

self.not\_met --> False if the criterea is met

"""

#max 10 iterations

if i > 3:

self.not\_met = False

#minimum self.S\_hat to be under .001m

not\_under = False

for key in self.S\_hat:

if abs(key[0,0]) > .0001:

#this means the criterea was not met for atleast one of the unknowns

not\_under = True

if not not\_under:

#then all things were under .0001m in change and therefore the criterea was met

self.not\_met = False

def covariance(self):

"""

Desc:

Initialized covariance matrix based on observation standard deviations

Input:

Output:

self.Cl

"""

self.Cl = mat(np.zeros((self.n, self.n)))

for i in range(0,self.n):

self.Cl[i,i] = self.errs[i]\*\*2

def design(self):

"""

Desc:

Set up overall design matrix

Input:

Output:

self.A

"""

self.A = mat(np.zeros((self.n, self.u)))

temp = []

for model in self.models:

temp.append(model.A)

self.A = np.vstack(temp)

def n\_mat(self):

"""

"""

self.N = mm(t(self.A),mm(self.P,self.A))

def cx\_mat(self):

"""

"""

self.Cx = inv(self.N)

def w\_mat(self):

"""

"""

#adds constants and unknowns together and solves for values

self.w = mm(self.A,LS.x\_0) - self.obs

#\_\_\_\_\_\_\_\_\_\_\_\_for non linear this will need to change\_\_\_\_\_\_

def u\_mat(self):

"""

"""

self.v = t(self.A,mm(self.P,self.w))

def correction(self):

"""

"""

self.S = -mm(inv(self.N),mm(t(self.A),mm(self.P,self.w)))

def obs\_0(self):

"""

Desc:

Assembles l\_obs from each matrix

Input:

Output:

self.l\_0 constructed

"""

self.l\_0 = mat(np.zeros((self.n, 1)))

temp = []

for obs in self.models:

temp.append(obs.l\_0)

self.l\_0 = np.vstack(temp)

def update\_values(self):

"""

Desc:

Updates x\_0 and design and l\_0

Input:

Uses most recent x\_hat value

Output:

none:

"""

#update models

for model in self.models:

#model.x\_0 = self.x\_0

model.obs\_0()

#update design matrix

model.set\_design()

#update within network

self.design()

self.obs\_0()

## LeastSquares.py

from numpy import transpose as t

from numpy import matrix as mat, matmul as mm

import math as m

import numpy as np

import pandas as pd

from Tools import Tools

class LS(Tools):

"""

Holds the universal values needed to integrate the different LS adjustments into one

"""

x\_0 = []

def \_\_init\_\_(self, file\_name = "coords.txt", debugging = False):

"""

Desc:

reads in the list of knowns and unknowns and assigns their values. Will construct design matrix, etc. based off of these

Input:

file\_name where the knowns and unknowns are defined

debugging, T/F. If true then more printing of stuff happens

Output:

sets up u\_list (predefined in here)

sets up number of unknowns (self.u)

"""

#brings in the tool files for use

Tools.\_\_init\_\_(self)

self.debugging = debugging

self.file\_name = file\_name

self.read\_2D()

def read\_2D(self):

"""

Desc:

reads in the 2D set of points and assigns values

expects format of [name easting northing known/unknown]

more specifically: [Point X[m] Y[m] Known[n]/Unknown[u]]

Input:

self.file\_name

Output:

self.u\_list (string list of unknown)

self.x\_0 (initial guesses of unknowns)

self.c (constant values of knowns)

self.datums (string list of knowns)

self.u # of unknowns

"""

df = pd.read\_csv(self.file\_name, sep = ' ')

#currently only formatted for 2D

self.u\_list = []

LS.x\_0 = []

self.c = []

#pretty sure datums aren't actually used

self.datums = []

#assign values

for index, row in df.iterrows():

#check if known or unknown

if row[3] == "u":

#unknown name

self.u\_list.append(row[0]+"\_E")

self.u\_list.append(row[0]+"\_N")

#add unknown values in order of x, y

LS.x\_0.append(row[1])

LS.x\_0.append(row[2])

else: #then they are "n" --> knowns

#known name

self.datums.append(row[0]+"\_E")

self.datums.append(row[0]+"\_N")

#add known values in order of x, y

self.c.append(row[1])

self.c.append(row[2])

LS.x\_0 = t(mat(LS.x\_0))

self.c = t(mat(self.c))

self.u = len(self.u\_list)

def find\_col(self, dimension, point\_name, li = "u"):

"""

Desc:

returns the column index of the desired points

expects 'n' for known and 'u' for unknown

\*\*all values must be in caps\*\*

Input:

u\_list, list of strings of "pointname\_dimension"

dimension, string either "N", "E", "H"

Output:

integer value of the column to place the value

in the desired design matrix

"""

if li == "u" :

li = self.u\_list

else:

li = self.datums

index = 0

for key in li:

#split the key into point name and dimension

temp\_name = key.split('\_')[0]

temp\_dimension = key.split('\_')[1]

if (point\_name == temp\_name and dimension == temp\_dimension):

return index

else:

index = index + 1

#debugging stuff

if self.debugging:

print(point\_name + " Could not be found")

return -1

## Distance.py

from numpy import matrix as mat, matmul as mm

import math as m

import numpy as np

import pandas as pd

from LeastSquares import LS

class Distance(LS):

"""

"""

def \_\_init\_\_(self, df\_name = "distances.txt"):

"""

Desc:

reads in the distance observations and preps that point of the LSA

Input:

df\_name

dimension\_word =

dimension\_symbol = "H", can be switched to "E", "N"

Output:

self.obs type: matrix: observation matrix

"""

LS.\_\_init\_\_(self)

self.df\_name = df\_name

self.read\_distance()

self.set\_obs()

self.set\_errors()

self.set\_design()

self.obs\_0()

def read\_distance(self):

"""

Desc:

reads in the distance stuff for a 2D

Input:

Output:

self.d\_word for the observations

self.d\_error for the stddev column

self.df of all info [From To Distance[m] StDev[m]]

"""

self.d\_word = "Distance[m]"

self.d\_error = "StDev[m]"

self.d\_symbol = "E"

self.df = pd.read\_csv(self.df\_name, sep = ' ')

def set\_obs(self):

"""

Desc:

sets up the observation matrix from the distance observations.

Input:

Output:

self.obs

self.n, number of observations

"""

self.obs = mat(self.df[self.d\_word]).transpose()

self.n = len(self.df[self.d\_word])

def set\_design(self):

"""

Desc:

initializes the design matrix with 0's, 1's and -1's

Input:

Output:

self.A, type matrix

"""

#set it up as just zeros

self.A = mat(np.zeros((self.n, self.u)))

#get the from and tos ready to be accessed

froms = self.df["From"].to\_list()

tos = self.df["To"].to\_list()

#making these variables always accessible so that no errors arrise (don't worry they aren't assigned if a partial derivative is meant for it)

e\_from = 0

n\_from = 0

e\_to = 0

n\_to = 0

#set placeholder

i = 0

while(i < self.n):

#this is for figuring out if its a known or unknown\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#the extra code here was incase approx values were needed to additionally populate the design matrices

from\_const = False

to\_const = False

#find columns to place values

#picks the dimension symbol to search for

from\_col = self.find\_col(self.d\_symbol, froms[i])

if from\_col == -1:

#honestly this is only if we were to do something with the partial of the datums but

#we don't so ignore this and the next similar if statement, other than setting the True value is important

from\_col = self.find\_col(self.d\_symbol, froms[i], li = "datums")

#then it is a datum

#sets easting and northing values from the froms

e\_from = self.c[from\_col,0]

n\_from = self.c[from\_col+1,0]

from\_const = True

else:

#sets easting and northing values from the froms

e\_from = LS.x\_0[from\_col,0]

n\_from = LS.x\_0[from\_col+1,0]

to\_col = self.find\_col(self.d\_symbol, tos[i])

if to\_col == -1:

to\_col = self.find\_col(self.d\_symbol, tos[i], li = "datums")

#then it is a datum

#to\_num = self.c[to\_col,0]

e\_to = self.c[to\_col,0]

n\_to = self.c[to\_col+1,0]

to\_const = True

else:

#set the easting and northing values of the

e\_to = LS.x\_0[to\_col,0]

n\_to = LS.x\_0[to\_col+1,0]

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

if self.debugging:

print("E\_from: "+str(e\_from)+" N\_from:"+str(n\_from))

print("E\_to: "+str(e\_to)+" N\_to: "+str(n\_to))

dist = m.sqrt((e\_from-e\_to)\*\*2+(n\_from-n\_to)\*\*2)

#this is where the values are assigned\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#it is important that files are formatted as X and then Y so that we can find the X column

#and autopopulate the Y column next to it

if not from\_const:

self.A[i,from\_col] = (e\_from - e\_to)/dist

self.A[i,from\_col+1] = (n\_from - n\_to)/dist

if not to\_const:

#self.A[i,to\_col] = delta + 1

self.A[i,to\_col] = -(e\_from - e\_to)/dist

self.A[i,to\_col+1] = -(n\_from - n\_to)/dist

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

i = i + 1

def set\_errors(self):

"""

Desc:

sets up the errors in an n,1 matrix

Input:

Output:

self.errs

"""

self.errs = mat(self.df[self.d\_error]).transpose()

def obs\_0(self):

"""

desc:

Sets up self.l\_0 (extimated observations)

Used for finding the current misclosure

"""

#set it up as just zeros

self.l\_0 = mat(np.zeros((self.n, 1)))

#get the from and tos ready to be accessed

froms = self.df["From"].to\_list()

tos = self.df["To"].to\_list()

#making these variables always accessible so that no errors arrise (don't worry they aren't assigned if a partial derivative is meant for it)

e\_from = 0

n\_from = 0

e\_to = 0

n\_to = 0

#set placeholder

i = 0

while(i < self.n):

#this is for figuring out if its a known or unknown\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#the extra code here was incase approx values were needed to additionally populate the design matrices

from\_const = False

to\_const = False

#find columns to place values

#picks the dimension symbol to search for

from\_col = self.find\_col(self.d\_symbol, froms[i])

if from\_col == -1:

#honestly this is only if we were to do something with the partial of the datums but

#we don't so ignore this and the next similar if statement, other than setting the True value is important

from\_col = self.find\_col(self.d\_symbol, froms[i], li = "datums")

#then it is a datum

#sets easting and northing values from the froms

e\_from = self.c[from\_col,0]

n\_from = self.c[from\_col+1,0]

from\_const = True

else:

#sets easting and northing values from the froms

e\_from = LS.x\_0[from\_col,0]

n\_from = LS.x\_0[from\_col+1,0]

to\_col = self.find\_col(self.d\_symbol, tos[i])

if to\_col == -1:

to\_col = self.find\_col(self.d\_symbol, tos[i], li = "datums")

#then it is a datum

#to\_num = self.c[to\_col,0]

e\_to = self.c[to\_col,0]

n\_to = self.c[to\_col+1,0]

to\_const = True

else:

#set the easting and northing values of the

e\_to = LS.x\_0[to\_col,0]

n\_to = LS.x\_0[to\_col+1,0]

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dist = m.sqrt((e\_from-e\_to)\*\*2+(n\_from-n\_to)\*\*2)

#this is where the values areassigned\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#self.A[i,to\_col] = delta + 1

self.l\_0[i,0] = dist

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

i = i + 1

## Angle.py

from numpy import matrix as mat, matmul as mm

from numpy import transpose as t

import math as m

import numpy as np

import pandas as pd

from LeastSquares import LS

class Angle(LS):

"""

"""

def \_\_init\_\_(self, df\_name = "angles.txt"):

"""

Desc:

reads in the anglular model data, expects format [From At To Degrees Minutes Seconds StDev[sec]]

Input:

df\_name

dimension\_word = "Height", can be switched to "Easting" or "Northing"

dimension\_symbol = "H", can be switched to "E", "N"

Output:

self.obs type: matrix: observation matrix

"""

LS.\_\_init\_\_(self)

self.df\_name = df\_name

self.read\_angle()

self.set\_obs()

self.set\_errors()

self.set\_design()

self.obs\_0()

def read\_angle(self):

"""

Desc:

reads in the distance stuff for a 2D

Input:

Output:

self.d\_word for the observations radians

self.d\_error for the stddev column

self.df of all info [From To Distance[m] StDev[m]]

"""

self.df = pd.read\_csv(self.df\_name, sep = ' ')

#Switch DMS to Radians\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.df = pd.read\_csv('angles.txt', sep = ' ')

deg = np.array(self.df["Degrees"].to\_list())

mins = np.array(self.df["Minutes"].to\_list())

sec = np.array(self.df["Seconds"].to\_list())

degree = deg + mins/60 + sec/3600

radians = np.radians(degree)

#drop DMS

self.df = self.df.drop(columns = ["Degrees", "Minutes", "Seconds"])

#add degrees

self.df["Radians"] = radians

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.d\_word = "Radians"

#Switch second error to radian error \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

self.df["StDev[rad]"] = np.radians(np.array(self.df["StDev[sec]"])/3600)

#self.df = self.df.drop(columns = ["StDev[sec]"])

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#now formatted [From At To Radians StDev[rad]]

self.d\_error = "StDev[sec]"

self.d\_symbol = "E"

def set\_obs(self):

"""

Desc:

sets up the observation matrix from the heights

Input:

Output:

self.obs

self.n, number of observations

"""

#switch from

self.obs = mat(self.df[self.d\_word]).transpose()

self.n = len(self.df[self.d\_word])

def set\_design(self):

"""

Desc:

initializes the design matrix with 0's, 1's and -1's

Input:

Output:

self.A, type matrix

"""

#set it up as just zeros

self.A = mat(np.zeros((self.n, self.u)))

#get the from and tos ready to be accessed

angles = self.df[self.d\_word].to\_list()

ats = self.df["At"].to\_list()

froms = self.df["From"].to\_list()

tos = self.df["To"].to\_list()

#print(ats)

#print(froms)

#print(tos)

w = 0

#set placeholder

i = 0

e\_from = 0

n\_from = 0

e\_to = 0

n\_to = 0

e\_at = 0

n\_at = 0

while(i < self.n):

#this is for figuring out if its a known or unknown\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#the extra code here was incase approx values were needed to additionally populate the design matrices

from\_const = False

to\_const = False

at\_const = False

#find columns to place values

#picks the dimension symbol to search for

from\_col = self.find\_col(self.d\_symbol, froms[i])

if from\_col == -1:

#honestly this is only if we were to do something with the partial of the datums but

#we don't so ignore this and the next similar if statement, other than setting the True value is important

from\_col = self.find\_col(self.d\_symbol, froms[i], li = "datums")

#then it is a datum

e\_from = self.c[from\_col,0]

n\_from = self.c[from\_col+1,0]

from\_const = True

else:

#sets easting and northing values from the froms

e\_from = LS.x\_0[from\_col,0]

n\_from = LS.x\_0[from\_col+1,0]

to\_col = self.find\_col(self.d\_symbol, tos[i])

if to\_col == -1:

to\_col = self.find\_col(self.d\_symbol, tos[i], li = "datums")

#then it is a datum

e\_to = self.c[to\_col,0]

n\_to = self.c[to\_col+1,0]

to\_const = True

else:

#set the easting and northing values of the

e\_to = LS.x\_0[to\_col,0]

n\_to = LS.x\_0[to\_col+1,0]

at\_col = self.find\_col(self.d\_symbol, ats[i])

if at\_col == -1:

at\_col = self.find\_col(self.d\_symbol, ats[i], li = "datums")

#then it is a datum

e\_at = self.c[at\_col,0]

n\_at = self.c[at\_col+1,0]

at\_const = True

else:

#set the easting and northing values of the

e\_at = LS.x\_0[at\_col,0]

n\_at = LS.x\_0[at\_col+1,0]

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#constants for this line

#print("======================")

#print(e\_from)

#print(n\_from)

#print(e\_at)

#print(n\_at)

#print(e\_to)

#print(n\_to)

#print("======================")

#this is where the values are assigned\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#it is important that files are formatted as X and then Y so that we can find the X column

#and autopopulate the Y column next to it

if not at\_const:

#self.A[i,to\_col] = delta + 1

#(1)

self.A[i,at\_col] = (((n\_to - n\_at)/((e\_to-e\_at)\*\*2+(n\_to-n\_at)\*\*2))-((n\_from-n\_at)/((e\_from-e\_at)\*\*2+(n\_from-n\_at)\*\*2)))

#(2)

self.A[i,at\_col+1] = ((-(e\_to - e\_at)/((e\_to-e\_at)\*\*2+(n\_to-n\_at)\*\*2))+((e\_from-e\_at)/((e\_from-e\_at)\*\*2+(n\_from-n\_at)\*\*2)))

if not from\_const:

#(3)

self.A[i,from\_col] = ((n\_from-n\_at)/((e\_from-e\_at)\*\*2+(n\_from-n\_at)\*\*2))

#(4)

self.A[i,from\_col+1] = (-(e\_from-e\_at)/((e\_from-e\_at)\*\*2+(n\_from-n\_at)\*\*2))

if not to\_const:

#self.A[i,to\_col] = delta + 1

#(5)

self.A[i,to\_col] = (-(n\_to-n\_at)/((e\_to-e\_at)\*\*2+(n\_to-n\_at)\*\*2))

#(6)

self.A[i,to\_col+1] = ((e\_to-e\_at)/((e\_to-e\_at)\*\*2+(n\_to-n\_at)\*\*2))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

i = i + 1

def set\_errors(self):

"""

Desc:

sets up the errors in an n,1 matrix

Input:

Output:

self.errs

"""

#converts from arc seconds to radians

self.errs = np.radians(mat(self.df[self.d\_error]).transpose()/3600)

def omega(self, fro\_e, fro\_n, to\_e, to\_n):

"""

Calculates omega from

"""

fro = np.array([fro\_e, fro\_n])

to = np.array([to\_e, to\_n])

north = np.array([0,1])

r\_v = to - fro

self.wi = np.dot(r\_v / np.linalg.norm(r\_v), north / np.linalg.norm(north))

def obs\_0(self):

"""

desc:

Sets up self.l\_0 (extimated observations)

Used for finding the current misclosure

"""

self.l\_0 = mat(np.zeros((self.n, 1)))

#get the from and tos ready to be accessed

angles = self.df[self.d\_word].to\_list()

ats = self.df["At"].to\_list()

froms = self.df["From"].to\_list()

tos = self.df["To"].to\_list()

#print(ats)

#print(froms)

#print(tos)

w = 0

#set placeholder

i = 0

e\_from = 0

n\_from = 0

e\_to = 0

n\_to = 0

e\_at = 0

n\_at = 0

while(i < self.n):

#this is for figuring out if its a known or unknown\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#the extra code here was incase approx values were needed to additionally populate the design matrices

from\_const = False

to\_const = False

at\_const = False

#find columns to place values

#picks the dimension symbol to search for

from\_col = self.find\_col(self.d\_symbol, froms[i])

if from\_col == -1:

#honestly this is only if we were to do something with the partial of the datums but

#we don't so ignore this and the next similar if statement, other than setting the True value is important

from\_col = self.find\_col(self.d\_symbol, froms[i], li = "datums")

#then it is a datum

e\_from = self.c[from\_col,0]

n\_from = self.c[from\_col+1,0]

from\_const = True

else:

#sets easting and northing values from the froms

e\_from = LS.x\_0[from\_col,0]

n\_from = LS.x\_0[from\_col+1,0]

to\_col = self.find\_col(self.d\_symbol, tos[i])

if to\_col == -1:

to\_col = self.find\_col(self.d\_symbol, tos[i], li = "datums")

#then it is a datum

e\_to = self.c[to\_col,0]

n\_to = self.c[to\_col+1,0]

to\_const = True

else:

#set the easting and northing values of the

e\_to = LS.x\_0[to\_col,0]

n\_to = LS.x\_0[to\_col+1,0]

at\_col = self.find\_col(self.d\_symbol, ats[i])

if at\_col == -1:

at\_col = self.find\_col(self.d\_symbol, ats[i], li = "datums")

#then it is a datum

e\_at = self.c[at\_col,0]

n\_at = self.c[at\_col+1,0]

at\_const = True

else:

#set the easting and northing values of the

e\_at = LS.x\_0[at\_col,0]

n\_at = LS.x\_0[at\_col+1,0]

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#constants for this line

#print("======================")

#print(e\_from)

#print(n\_from)

#print(e\_at)

#print(n\_at)

#print(e\_to)

#print(n\_to)

#print("======================")

#this is where the values are assigned\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#it is important that files are formatted as X and then Y so that we can find the X column

#and autopopulate the Y column next to it

#this is where the values are generated\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#it is important that files are formatted as X and then Y so that we can find the X column

#and autopopulate the Y column next to it

ang = m.atan((n\_to-n\_at)/(e\_to-e\_at)) - m.atan((n\_from-n\_at)/(e\_from-e\_at))

if ang < 0:

ang = ang + m.pi

#this is where the values are assigned\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#self.A[i,to\_col] = delta + 1

self.l\_0[i,0] = ang

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

i = i + 1

## PostAdjustmentTester.py

from numpy import transpose as t

from numpy import matrix as mat, matmul as mm

import matplotlib as plt

from numpy import linalg as lin

from numpy.linalg import inv

import math as m

import numpy as np

import pandas as pd

from LeastSquares import LS

from Level import Delta

from Tables import Tables

from scipy import stats as st

from scipy.stats import t as stu

from scipy.stats import chi2

class PostAdjustmentTester(Tables):

"""

Desc:

Assumes that the LSA has been conducted and outputs results for post adjustment tests

"""

def \_\_init\_\_(self):

"""

Desc:

Figuring out if we need to take in matrices or if we'll just inherit the class and assume that they're build

"""

Tables.\_\_init\_\_(self)

def global\_a\_posteriori(self, alpha = .05):

"""

Desc:

Tests the statistical sifnigicance of the aposteriori to a priori variance factor

Input:

alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values, otherwise they won't be found :-)

self.u: # of unknowns

self.n: # of observations

self.a\_post: final computed a posteriori variance factor

self.apriori: initial apriori variance factor

Output:

Prints the output and respective indication

"""

#set up DOF (r)

self.r = self.n - self.u

#retrieves dataframe of chi values for our respective DOF

ch\_df = self.x\_2()

low = ch\_df[alpha][0]

high = ch\_df[1-alpha][0]

y = (self.r \* self.a\_post\*\*2)/self.apriori\*\*2

#if fails this check then there is an indication that the residuals or math model may be off

print("{} tested with chi\_square boundries of {} and {}".format(y, low, high))

if y > low and y < high:

print("Global A-Posteriori Variance Factor Test passes at a {} confidence level".format((1 - alpha)\*100))

print("There is no indication for errors within residual or the math models")

else:

print("Global A-Posteriori Variance Factor Test \*\*failed\*\* at a {}% confidence level".format((1 - alpha)\*100))

print("There is indication that errors exsist within residual or the math models")

def significance\_estimated\_param(self, alpha = .05):

"""

Desc:

Determines whether there is statistical signifiance to beleive the final estimated value of parameters

Input:

alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values, otherwise they won't be found :-)

self.n: # of observations

self.x\_hat

self.u\_list: for labelling

self.Cx: for extracting std dev values of parameters

Output:

retrunds dataframe of values [Unknown Final Value Value Standard Deviation Test Value Indicated Significance Alpha Tested Confidence Level Test Bounds]

"""

#set up DOF (r)

self.r = self.n - self.u

high = stu.ppf(1.0 - alpha, self.r)

low = stu.ppf(alpha, self.r)

#final paramter values

xs = []

#unknown names in string format

self.u\_list

us = []

#list to store their signifiance as Signifiance or Not Significant

sig = []

#list to store the value that was checked

sig\_value = []

#list of standard deviation values

std = []

#test values

y = []

#confidence levels

conf = []

#confidence levels

alphs = []

#test bounds

bounds = []

for i in range(0,self.u):

std.append(m.sqrt(self.Cx[0,0]))

y.append((self.x\_hat[i]/std[i])[0,0])

if y[i] > low and y[i] < high:

#if fails then there IS statistical significance

sig.append("No")

else:

sig.append("Yes")

xs.append(self.x\_hat[i][0,0])

us.append(self.u\_list[i])

conf.append((1-alpha)\*100)

alphs.append(alpha)

bounds.append(str([low, high]))

#to store values in a dictionary before conversion to dataframe

dict\_list = {

"Unknown": xs,

"Final Value": us,

"Value Standard Deviation": std,

"Test Value": y,

"Indicated Significance": sig,

"Alpha Tested": alphs,

"Confidence Level": conf,

"Test Bounds": bounds

}

#return dict\_list

return pd.DataFrame.from\_dict(dict\_list)

def semi\_global\_residuals(self, alpha = .05):

"""

Desc:

Conducts the semi global test on residuals, also known as the gooness-of-fit or normality test on residuals

Input:

self.r\_hat: residuals

alpha = .05: to find confidence level

self.Cr: extracting std of residuals

self.n: number of observations

Output:

prints whether the test passed and the reccomended interpretation

"""

#normalize residuals

norm\_r = []

for i in range(0,self.n):

norm\_r.append(self.r\_hat[i,0]/m.sqrt(self.Cr[i,i]))

#number of bins

M = round(m.sqrt(self.n))

counts, bins = np.histogram(norm\_r, bins = M)

#compute estimated number of residuals per bin

e = []

for i in range(M):

#get probability of total bin

p\_start = st.norm.cdf(bins[i])

p\_end = st.norm.cdf(bins[i+1])

p = p\_end - p\_start

#append total number of expected residuals

e.append(p\*self.n)

#compute X\_2 for each bin

chis = []

for i in range(M):

chis.append((e[i]-counts[i])\*\*2/e[i])

#sum all chis for test statistic y

y = sum(chis)

#conduct statistical test

dof = M - 1

prob = 1 - alpha

chi = chi2.ppf(prob, dof)

print("{} tested with chi\_square of {} ".format(y, chi))

if y > chi:

print("The Semi-Global, goodness-of-fit test on the residuals \*\*Failled\*\*")

print("There is a sign that either there are outliers or the functional model was not appropriate for the data set")

else:

print("The Semi-Global, goodness-of-fit test on the residuals \*\*Passed\*\*")

print("There is no sign of outliers or functional model errors")

#plt.hist(norm\_r, m)

def blunder\_detection(self, alpha = .01):

"""

Desc:

Conducts the local test on the residuals, aka blunder detection

Input:

alpha = .01: for 99% confidence of a blunder

self.Cr: for extracting std of residuals

self.r\_hat: for extracting residuals

Output:

Returns a dataframe with columns ["Observation", "Outlier", "Test Value", "Test Bounds"]

"""

#statistical test values

low = st.norm.ppf(alpha/2)

high = st.norm.ppf(1-alpha/2)

#normalize residuals (test statistic)

y = []

#list of Yes or No outliers

outlier = []

#confidence levels

conf = []

#observations

observations = []

#test bounds

bounds = []

for i in range(0,self.n):

#for DF

observations.append(i)

bounds.append(str([low, high]))

conf.append((1-alpha)\*100)

y.append(self.r\_hat[i,0]/m.sqrt(self.Cr[i,i]))

if y[i] > low and y[i] < high:

#passes test --> not an outlier

outlier.append("No")

else:

outlier.append("Yes")

dic = {

"Observation": observations,

"Outlier": outlier,

"Confidence Level": conf,

"Test Value": y,

"Test Bounds": bounds

}

return pd.DataFrame.from\_dict(dic)